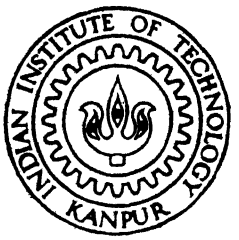
An approach to testing message centric object-oriented specifications

by
R. K. SHREEVASTAVA



DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY KANPUR

April, 1997

An approach to testing message centric object-oriented specifications

A Thesis Submitted

in Partial Fulfillment of the Requirements

for the Degree of

Master of Technology

by

R. K.Shreevastava

to the

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

April 1997

24-APR 1997 / CSE

JR

Ins. No. A 123309

CSE-1997-M-SHR-APP

CERTIFICATE

This is to certify that the work contained in the thesis entitled "An approach to testing message centric object-oriented specifications" by "R.K.Shreevastava' has been carried out under my supervision and that this work has not been submitted elsewhere for a degree.

Prof. Harish Karnick
Computer Science & Engineering Department
Indian Institute of Technology, Kanpur.

pril, 1997

Acknowledgements

I am grateful to my thesis supervisor, Dr. H. Karnick, who constantly guided, motivated and helped me to complete my thesis. His continuous encouragements, never let me lose my confidence. It was very educative and satisfying for me to have worked under Dr. H. Karnick.

I would like to thank my friends Warsi, Milind, Samir and D Srinivas for lending me their valuable time alongwith some useful suggestions, for my thesis. Specially Samir and D Srinivas who have helped a lot during my thesis.

Abstract

We propose an approach for testing and validating message centric specifications. Testing is based on use cases, and does not attempt exhaustive testing. Messages contain pre- and post- conditions, based on which an object sends or receives a message. These conditions represent, the state of an object. An object will not send a message till the precondition is satisfied. Similarly an object will not receive a message till the postcondition, associated with the message, is satisfied.

We test the system and generate test cases for the system, using these pre- and post- conditions. Testing and test case generation is done in two steps. Firstly, all use cases are generated. During which a table is formed, where all messages associated with a particular class is stored under that class. Secondly testing is done and test cases generated by considering the pre- and post-condition of all messages for a particular use case. These test cases can then be used to validate the system so developed.

Contents

1 INTRODUCTION 4

1.1 Statement of problem 4

1.2 Testing 6

1.3 Testing in a message based specification systems 6

1.4 Overview of thesis 7

2 Extension of previous work 8

2.1 About message based specification system 8

2.1.1 Notation 10

2.1.2 The development process 17

2.2 Rapid prototyping for message centric based specification system . . 20

2.2.1 Symbol table manager (STM) 20

3 Design 22

3.1 Usecase generation 24

3.1.1 Send message 25

3.1.2 Do subprocess 27

3.1.3 Invoke menu 27

3.1.4 Invoke fill in 27

3.1.5 Select menu item 28

3.2 Testcases 28

3.2.1 Received message id 29

3.2.2 Received resultof message 30

3.2.3 Done subprocess 32

3.2.4 Constraints 33

4 Implementation **34**

4.1 Use_Case_Msg 34

4.2 newClass 35

4.3 Start_ptr_table 35

4.4 Implementation 35

5 CONCLUSION **36**

5.1 Conclusion 36

5.2 Results 36

5.3 Possible extensions 37

A Pseudocodes **38**

A.1 Pseudocode for usecase generation 38

A.2 Pseudocode for testing and testcase generation 40

B Example of DOAA office automation **47**

List of Figures

1 Creating usecase as sequence of messages 23
2 Example of a usecase 24
3 Checking for inconsistency 31

Chapter 1

INTRODUCTION

1.1 Statement of problem

There are many methodologies using which large and complex software can be built. Out of these methodologies object oriented methodologies are quite effective in managing such large and complex software. In a system which is built using the OO approach, static structure of the software is described by the object model. These objects then interact with each other to implement the system.

A message based specification system for the object oriented software was developed [Sar96], which effectively captures the dynamics of the system using a formal notation. In message centric specification system, the objects of the object model send messages amongst themselves to implement the functionality of the system. These messages constitute the message model of the system, which captures the dynamics of the system. In the message model we have the pre- and post- conditions, which lay down constraints on the state of the objects.

Precondition is the condition on the state of the sender before sending the message. Thus, an object will not send a message till the condition is satisfied. Similarly, a postcondition is the condition on the state of the receiver before receiving the message. Hence, an object (receiver) will not receive a message till postcondition is satisfied. For example in a usecase, class DOAA sends the message "give_rule" to class RULE_BOOK. The precondition of this message is that DOAA class (i.e.

the sender class, as the precondition is associated with the sender class) should have received "result_of_compute_cpi". So till the time DOAA class does not receive result_of_compute_cpi, it will not send the message "give_rule" to the class RULE_BOOK. Once the DOAA class receives the result_of_compute_cpi, it will send the message "give_rule" to class RULE_BOOK.

In this thesis we propose an approach to testing and testcase generation using these pre- and post- conditions. We would be checking for inconsistency in the specifications (for the system being developed using the message centric specification system) and also generating testcases. If we try to generate testcases for all possible messages an object will send or receive, then it would become intractable. Hence, we try to generate testcases based on the usecases. Thus only those messages, which are used in the usecase will be used for generating the testcases. This would limit our scope for generating the testcases.

We are using usecases for testing and generating the testcases. A usecase is full trace of one complete interaction between a user and the system. Thus all usecases together represent the various ways in which a user will use the system or interact with the system. Hence usecase based testing and testcase generation is the basic aim of our thesis.

Our problem is, therefore, testing and generating testcases for a system (built using message centric specification system), which is based on usecases only. An object may send/ receive many messages, but only those messages which figure in usecases are used for testing and testcase generation. This helps in saving :

- programmer time
- effort
- machine time
- economy of the project

1.2 Testing

In software construction, errors can be introduced at any stage during development. Generally the errors in specification and design are difficult to detect. These errors are reflected in the code. Since code is the only executable part of software whose behaviour can be observed, testing of software removes the errors remaining from earlier phases. Testing is crucial for ensuring quality and reliability of the software. During testing, the program to be tested is executed with a set of testcases and the output of the program for testcases is evaluated to determine if the program is performing as it is expected to.

Test activity is divided into verification and validation. Our thesis concerns validating the system. Testing object oriented software can be carried out at various levels. Unit testing is the lowest level of testing and is normally done by the developer himself. Unit tests are performed for classes, blocks and service packages. The next level of testing involves integration testing. The purpose of integration testing is to test whether the different units that have been developed are working together properly. It is performed by testing usecases one at a time, both from internal and external viewpoint. By external viewpoint we mean the interaction between the user and the system. And by internal viewpoint we mean the interaction amongst the various units/modules of the system. Each usecase then corresponds to a set of specifications. Finally in system testing, when all usecases have been tested properly, the entire system is tested as a whole, i.e. all the usecases are assumed to be running in parallel to check for the proper functioning of the system.

1.3 Testing in a message based specification systems

In a message based specification system, the message model consists of preconditions and postconditions. The conditions are constraints on the state of the sender/receiver class before sending/receiving messages. These condition are useful in testing purpose. As these conditions are constraint on the state of the object,

these constraints would then act as testcases for the object for a particular usecase. Once testing and testcase generation is completed the system can be validated. Validation is part of the testing, wherein we check whether the system has been built as per the specifications. In our thesis, during testing we are checking for inconsistency in the specification and testcases can be used to determine the correct functioning of the system. However, validation is a different process in itself and is not dealt with in our thesis.

1.4 Overview of thesis

In this thesis we present a tool to generate testcases from the pre- and post- condition of the message model, based on the particular usecase. Our tool can be used for validation of the software, which other testing process do not do.

The rest of the thesis is organized as : Chap 2 gives an introduction to the message based specification system. It basically gives the notation used and the process to be followed to arrive at the model. Chap 3 describes the design of the testcase generator. Chap 4 describes the implementation details where only the important structures have been described. Chap 5 gives the concluding remarks and possible extension. Appendix A gives the pseudocode for usecase and testcase generation. Appendix B gives an example of DOAA office automation which includes registration process, evaluation process and academic probation process.

Chapter 2

Extension of previous work

In this chapter we describe in brief the concept used in message based specification system, the notation used, and the development process in brief. Also described here is the on going work on the rapid prototyping tool for message centric based specification system. The description given here is very brief, for details refer to the original thesis [Sar96].

2.1 About message based specification system

In this method the system is viewed as a collection of objects communicating with each other. Each object has some data that constitute its state and some rules that tell when to send a message, when to receive a message and what to do when a message is received . As all these rules involve messages, the rules are specified with the message rather than with the object.

Below, we explain the terms that are used in this method are explained.

Object An Object is an abstract or real world entity which has state, behavior and identity, and communicates with other objects.

Most of the O-O methodologies give too much importance to the autonomous nature of an object that make the object an isolated entity. An object has both autonomous characteristics and interdependent characteristics. The attributes of an object are autonomous to it, i.e. depends on the nature of the objects.

But the state of the object depends on the interaction with other objects. As far as the software systems are concerned, objects do not exist in isolation.

State The state of an object at any instant consists of its knowledge about itself and its knowledge about the environment at that instant.

The knowledge about itself is represented by the values of the attributes of that object. The knowledge about the environment is represented by the messages it transacts.

Message A message forms the unit of communication between two objects.

The message is defined by the sender, receiver, message identifier and the arguments for the message. Also specified along with the message are Pre conditions (Constraints on the state of the sender to send the message), Post conditions (Constraints on the state of the receiver to receive the message) and Action to be performed by the receiver on receiving that message.

Process A Process is a sequence of events that happen to capture a meaningful real-world functionality.

Processes and objects are two central concepts of this method. In the first stage (requirements specification) there will be only processes but no classes or objects. In Analysis and Design stages there will be both classes, objects and processes. And in the implementation there will be only objects but no processes.

In requirements specification, processes are represented by plain English sentences. In Analysis and Design stages processes are represented in terms of messages and sub-processes, which again are represented in terms of messages.

Generic process and Generic message Generic process is the mechanism by which we can represent similar turn of events once and reuse it later. Similarly generic messages can be used when similar messages are being sent to different objects.

Sub process Sub process specification is useful to handle complexity. Large processes can be decomposed into sub processes. Generally, if there is one central object that interacts with several objects to realize an operation then we put these interactions into a sub process.

Usecase Usecase is a specific way of using the system by performing some parts of the functionality. Its a special sequence of related transactions performed by an user and the system in dialogue. Thus a complete collection of usecase would specify all the existing ways of using the system.

2.1.1 Notation

Object model

The object model is used to represent the static structure of the system. There are three kinds of objects in the system :entity objects, interface objects and control objects. Entity or data objects are used to store information. They usually exist in the problem domain and represent some real-world entity or concept. Interface objects are used to view data in entity objects. Control objects are used to collaborate between several entity and interface objects. The notation for entity and control objects is same and is different from notation for interface objects.

Class Specification This specification is valid for entity and control objects.

CLASS : Name of the class.

Name of the class is an identifier. Syntactically, an identifier begins with a letter [a-zA-Z] and contains any alphanumeric character (including "_"). This declaration begins declaration of a new class. All the items that follow this declaration are assumed to belong to this class. Only another "CLASS : xxx" or end-of-file will end the declaration of this class.

TYPE : ABSTRACT

This field is optional. if nothing is specified about the TYPE, then it will be taken as non-abstract data class. An abstract class is one which does not have

any instance. A class can be an abstract class only if it is a base class of an inheritance hierarchy.

INHERITS : class name [,class name s]

The list of classes from which this class inherits.

ATTR : attr name <type> , attr_name <type>

list of attributes each specified as attribute_name <type>. type can be any of the following :

- STRING, NUMBER, DATE.. etc. (i.e. scalar type)
- attr_decl (represents list)
- TABLE attributes list (a table)
- attributes_list (set of dissimilar but related items)

GENERALIZATION OF : class_name , class_name, ...

class_name are classes which inherit this class.

AGGREGATION OF :

Class name (multiplicity) .

PART OF : Class name (multiplicity)

Multiplicity is the number of part of objects of the given class in the aggregate object . Note that multiplicity and class names are specified for part-of objects also. This means that there can be shared aggregation.

For example, Department is collection of N number of system, N number of faculty members etc.. Student will belong to only one department, but a faculty member may belong to more than one department. Hence for student multiplicity is 1, and for the faculty member it is N.

CLASS : STUDENT
GENERALIZATION OF : UG_STUDENT

CLASS : DEPARTMENT
AGGREGATION OF : STUDENT (N), FACULTY (N)

CLASS : STUDENT
PART OF : DEPARTMENT (1)

CLASS : FACULTY
PART OF : DEPARTMENT (N)

RELATED WITH :

Related Class name (relation_name, multiplicity) Relation_name is the role the related object plays with respect to this object in the relation. Multiplicity specifies how many instances of the Related Class are related with one instance of this class.

Interface object specification CLASS NAME : Name of the interface class

TYPE : INTERFACE

MENU ITEMS :

id : "menu_identifier" name : "name of the menu" action : Action to be taken when this menu item is selected description : help message

Menu_identifier is used to uniquely identify this menu item. In addition to normal identifier syntax, "/" symbol can be used to denote the hierarchy of menu items. Name is the name of the menu item in more detail. This name is displayed to the user. Description is the message to be displayed when the user asks for help about this menu item.

Action will be one of

- send message msg_id [of process_id [to object]]
- do sub_process subproc_id
- invoke_menu menu_id
- select menu_item
- select fill_in_item

The actions are explained in Message specifications.

Fill_in items are generally used to represent data in forms and to take some temporary input data.

DISPLAY METHOD : These are used to specify logical display of screen with fill_in and menu items.

INHERITS :

GENERALIZATION OF :

Semantics of inheritance for interface classes differs slightly from that for normal classes. If interface class A inherits from interface class B, then A can use or replace some or all fill in items and display methods of B, and can also extend those menu items of B whose "actions" are invoke_menus.

RELATED WITH : CLASS (relation, multiplicity)

This has the same semantics as in normal class definitions. CLASS in RELATED WITH is used to declare which classes the interface object needs to attach to its fill_in items.

Message model

PROCESS: process_name

SUB PROCESS :: sub_process_name.

we divide something into sub processes when there is a central object that is gathering data from several objects to do certain operations.

GENERIC SUB PROCESS :: generic_sub_process (parameters)

INPUTS : parameters_varying_for_generic_sub_process.

The declaration of a set of messages begins with one of the above three process, sub process or generic sub process declarations.

FROM : Class name.

TO : Class name (constraint). Class name is the name of the class specified in the object model. Constraint can be

- condition on some attribute of the object. This is used to identify particular instance of the class.

- variable = class::attribute class::attribute1 = class::attribute2
- ALL, meaning all the instances of the class should be used for the purpose.
- ALL_RELATED, meaning all the instances receiver class related to this object.

HANDLE == ref
HANDLE == BACK

The above means that we explicitly give the handle. The handle identifies an object in the system. Every object will have an unique handle. In the context of a Database, handle is the primary key. When the handle is BACK, this means that the sender should not pick up the object handle from its Relations table, but should just send it, as a response to the message it received, to the corresponding object.

MSG_ID : message identifier (parameters).

The syntax of message identifier is same as that of class name in Class specification. The syntax of the parameters is the same as syntax of "list of attributes" as described in the Class specifications.

MSG_TYPE : type of the message.

Type of the message can be reply, shared or exclusive. Reply means the message is being sent as a reply to a message this object has previously received. The sender of the request message expects and waits for this reply message. Shared and Exclusive represent the security level of the message. If it is specified as exclusive, then only this object can send the message to the receiver. If it is specified as shared, then other objects can send this message to the receiver only if security level in that message specification is also specified as shared. shared is useful to restrict the senders to a set of objects.

MSG_COND :

Constraint that does not depend on the state of the sender, to send the message. This is used to send a message repeatedly (looping of the message) or

send a message depending on some condition which is not associated with the state of the object. For reasons of clarity, these message conditions surround the entire thread of messages which are invoked in a sequence as a result of issuing this message.

RESULT: reply to this message

The syntax of "result" is similar to that of list of attributes. When you specify the "result", it means that the sender "expects" some response to this message. This response is the result of action taken by the receiver of the message.

PRE_COND : condition on the state of the sender before sending the message

POST_COND : condition on the state of the receiver before receiving the message

It is the combination (and/or) of one or more of the following things.

- received msg_id [of process_id]
- received result_of msg_id [of process_id]
- sent msg_id [of process_id]
- attribute = value
- attribute1 = attribute2

The Pre condition is a constraint on the state of the sender to send the message. Post condition is a constraint on the state of the receiver to receive the message. By specifying different Post conditions for the same message exchanged between same two objects, we can have different actions depending on when the message is received. Other use of Post condition is to decide whether to accept or reject the message depending on its time of arrival. While coding, Pre condition is generally useful to position the function call. Pre and Post conditions are also important for testing purposes.

ACTION: action to be taken after this message has been received

Action can be any of the following

- send message msg_id [of process_id]

  For example, class STUDENT_INTERFACE sends the message "get_reg_form" to class STUDENT. The action associated with this message is send message "req_for_reg_form".

- do sub_process sub_proc_id (parameters)

  For example, class DOAA send message "notice_for_registration" to class

- send generic message msg_id

  Generic message as described earlier are similar messages sent to different objects.

- do generic sub_process process_id (parameters)

  For example, class DOAA_INTERFACE sends message "prepare_reg_notice" to class DOAA. The action associated with this message is to do generic subprocess "make_notice".

- select menu_item xxx

  For example, class DOAA_INTERFACE has two menu items under the menu id grades. These menu items are prepare_grades_sheets and send_grade_sheets. We select a particular menu item by this action.

  "select menu_item xxx" can be used only if the receiver is an Interface object.

GENERIC MESSAGE:

declaration of normal message, except that some fields will be empty.

VARIABLES IN MESSAGE

To specify the source of a variable (an argument to the Message or a variable in a PRE-COND etc.), we use the following :

- an attribute of sender variable.
- Sender receives it as an argument of some other message of this process!msg_id!variable .

- Sender receives it as an argument of some other message of some other process proc_id! msg_id_variable .
- Sender receives as a result of the message that was sent by it in this process proc_id! msg_id=variable.

If it is a loop variable (Eg. foreach loop etc.) nest_level variables. If the sender of the message is an Interface Object, then the interface object may use some data which it has got as temporary input. this data can be represented by variable::list. Internal values of message fields can referred by prefixing the field with $. This can be useful in generic processes.

2.1.2 The development process

The development process is a set of guidelines to identify effectively various components of the models described.

The system is built in the following phases : Specifying requirements, Analysis, Design and Implementation. Here we give what activities to do in each phase (except implementation).

Requirements Specification

Specify the system's requirements in terms of processes that the system consists of. Each process has to be expanded listing the sequence of events occuring in that process. While expanding a process, in addition to normal events, exceptional events should also be specified. Also try to identify various use-cases. As pointed out earlier, use case is a specific way of using the system by performing some part of the functionality. Thus, use cases help in identifying what is outside the system and what functionality the system should provide.

Analysis

The first thing to do in Analysis is to identify what kind of software architecture (or mixture of architectures) this system belongs to. This makes Analysis and Design easy as we will have knowledge of how similar systems have been

modeled in the past. This is very important particularly if we want to make re-use of design and code.

Initial object model

There are two kinds of identification in this method (be it identifying objects, attributes or messages). Identifying by intuition and identifying when necessary. When looking at a system, one can intuitively capture some obvious parts of the system. We represent these parts of the system in the Initial object domain model. Also, having some essential objects helps in starting with the message model.

In the initial object model, all the possible "obvious" classes, structures, attributes and services will be specified. Note that this need not be a comprehensive list of all objects in the system. Even if some are missed, they will be captured while specifying the message model.

We can specify the trivial interface objects also at this stage itself. These are interfaces corresponding to the human users of the end-system. Every possible user (generally roles with respect to the system) needs an interface to the system.

Analyze the Processes (Building message model and refining object model)

Specify each process as a sequence of messages. In the best case, each sentence in the process description can be translated into a message. Initially it is better to start with the simplest specification of a message : Sender, receiver and message_id. Because here we do not assume that process descriptions (given in the requirements specification phase) are unchangeable. As we can generate graphical feed-back even from this simple specification, it may help to correct requirements specification at an early stage. This (correction) does not cause any problem because the traceability between process descriptions and the initial message model is very high.

You may need to add some more objects to the system as the need arises while converting a process to messages. So, this will lead to discovery of

more objects (Application specific) and attributes. The two activities in this phase, identifying objects and identifying messages, are very much interleaved and interdependent. We can not specify any particular order regarding which activity is to be done first and which activity is to be done next. That is why both the activities are put in a single phase.

Message model and object model complement each other in system development. One may find that more objects are required to model the messages generated by processes/sub processes. This will add objects to the object model, thereby refining the object model. The new objects may help to refine the message model (by re-defining the boundaries of the messages).

Refining message model

Specify each message in more detail. i.e. giving Pre conditions, Post conditions, constraints etc. The message should be completely defined so that we identify its position (in control flow) independent of other messages.

Merging of Analysis models of Various processes

If there is division of labour in the analysis, i.e. if different sets of processes are analized by different groups of personnel, then there are bound to be some common objects and attributes that are discovered by more than one group. So, these have to be merged to form a single representation for the object model, and any inconsistencies have to be resolved. It is better to coordinate among the groups whenever there is a change to the object model as the modifications to the message models will be easy initially.

Design

In the Analysis, we describe what is to be done in the system. In the Design, we show how it is to be done. This "how" corresponds to the Implementation domain but not the problem domain.

In design we need not add any extra objects or messages to the specifications. We have to decide how to implement various components of the analysis models. Typically there are two phases of design. Designing at Macro level and

Designing at micro level.

Designing at macro level involves deciding overall architecture of the system : whether it has to be distributed or monolithic, if it is distributed, dividing the object model into Sub systems and allocating sub systems to Computers.

While designing at the micro level, we have to look at each object and see what all messages it receives and sends, and specifying algorithms for its private methods (generally "self" messages in the message model). Also, design a mechanism for storing and retrieving transient data that is sent along with messages.

2.2 Rapid prototyping for message centric based specification system

The rapid prototyping tool is a tool for executing the requirement specification as given by the user. It reads the requirement specification,and executes the system according to the specs. Thus enabling the user to only input the requirement specs and check the system being run. This tool has the following components : -

- Symbol table manager (STM)
- Interpreter
- Persistence manager

Out of, these components only the STM is relevant to our thesis, hence it is described in brief here. For more details and other components refer [Sri97] M.Tech thesis "A rapid prototyping tool for message centric specification system" which is being done concurrently.

2.2.1 Symbol table manager (STM)

The STM is the front end portion of the tool. It reads the requirement specification, parses it and creates and maintains tables. These tables consist of all the classes,

their attributes, functions, messages and their details. Once the STM is created, it is referred to and the details extracted during usecase generation portion of the testing and testcase generation. Description of how the STM is used during usecase generation is described in the next chapter.

Chapter 3

Design

In this chapter is described how the testcases are generated from the requirement specifications. The symbol table manager parses the requirement specification and maintains a table which has all the objects and messages passed amongst them. We use this symbol table manager in usecase generation, extensively. Testing and testcase generation is divided into two parts:-

- usecase generation
- testcases

In usecase generation we start from an interface class (these classes have menu items). Each menu item is taken as a starting point of a separate usecase. These menu items have a n action field which may be of the type send message. We extract the message id of this send message and lookup in the STM. Once a match is found, the action field of the found message is again checked. If its again of the type send message, the process of extracting message id, looking up in STM and finding a match is performed. Thus creating a sequence of message starting from a menu item. The usecase is complete when the message found in STM has no action field. (Refer figure for clarity)

In testing and testcase generation, we take each usecase. During usecase generation all messages being sent or received by a class is stored under that class. As

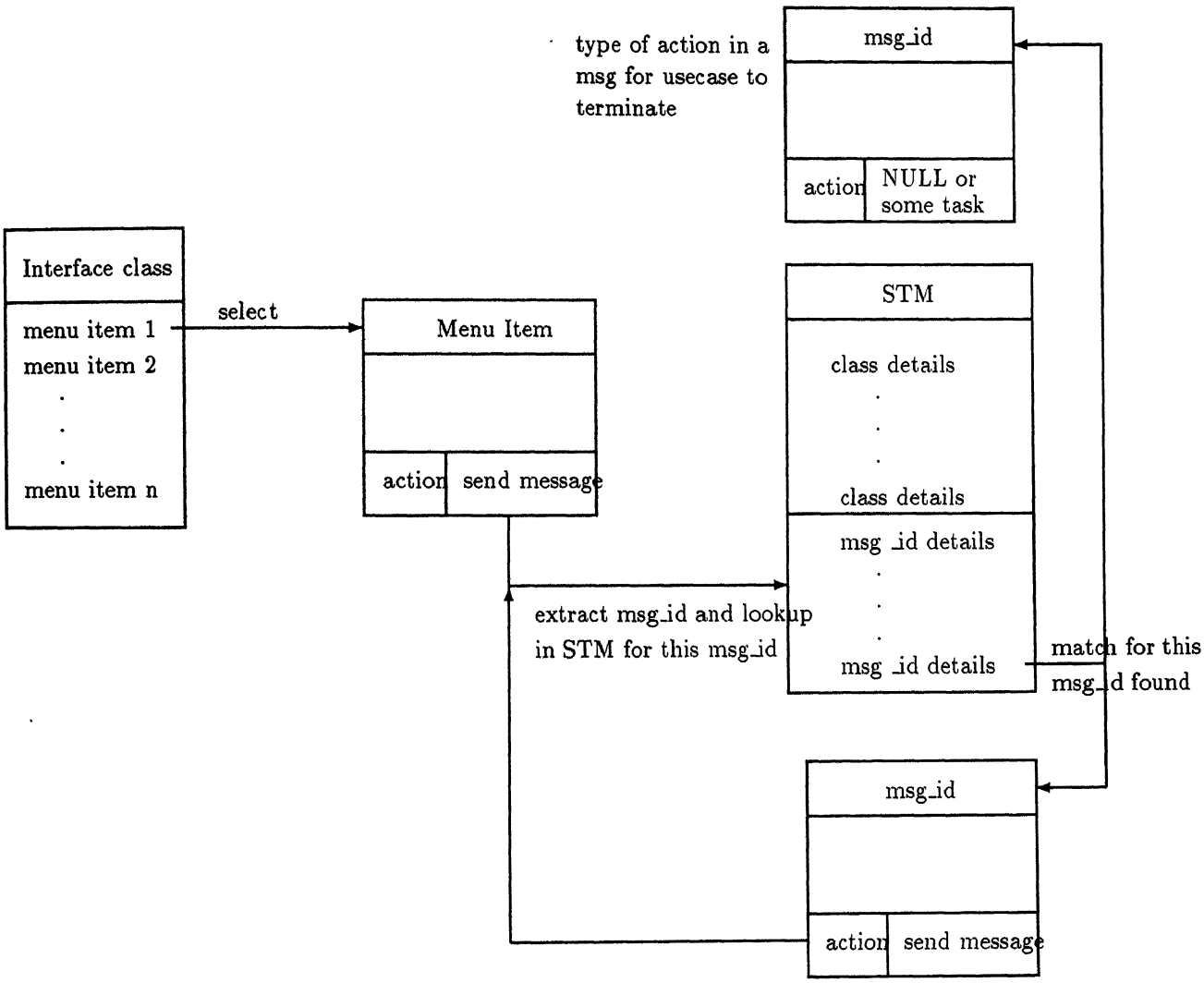


Figure 1: Creating usecase as sequence of messages

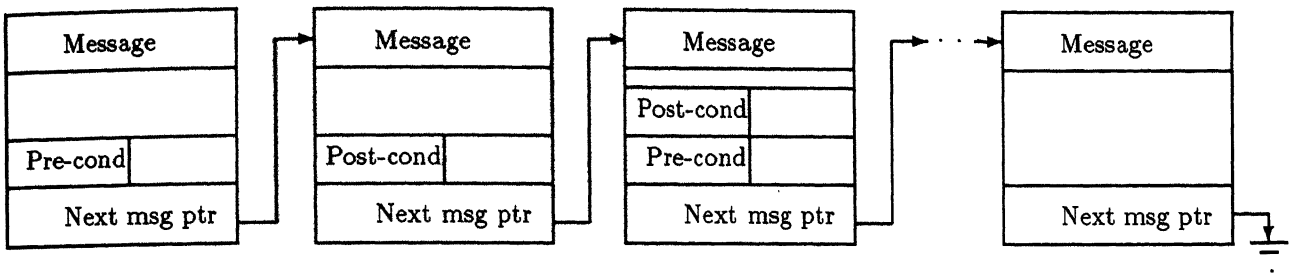


Figure 2: Example of a usecase

usecase is stored as linked list of messages, each message of the usecase under consideration is taken. And the pre- and post- condition of these messages are checked and testcases generated for this usecase, for the particular class.

3.1 Usecase generation

A usecase is full trace of one complete interaction between a user and the system. For example in DOAA office automation let us consider a usecase in evaluation process. The user selects the menu item send_grade_sheets to class STUDENT. The class STUDENT in response to this message stores the grade_sheets. Thus, in this usecase, the grade sheet is being given by the DOAA to the STUDENT. Thus, in a message based specification system we take the interface class as the starting point, as these are the classes the user uses to interact with the system. A tree traversal algorithm is used to access each and every menu item of the interface class.

Each interface class has a menu item field which identifies the name of the menu item and action field which describe the action to be taken when that particular menu item is chosen. We have considered each menu item to be a separate usecase. Thus each traversal through the menu tree to a menu item gives a separate usecase. Based on the type of action various usecase are generated. The different action are:-

- send message
- do sub process

- invoke menu item
- select fill in

An interface class consists of fill in items, apart from the menu items. These fill in items are invoked by select fill in type of action of menu item.

3.1.1 Send message

When the action of the menu item selected is to send a message of a particular process then the message is looked for in the symbol table manager. If the same message is not found, then the usecase is terminated there itself. For example, in the interface class DOAA_INTERFCAE there is a menu item "grades". This menu item has action to invoke another menu. The menu's invoked are

- prepare_grade_sheet
- send_grade_sheet_to_dept

We consider prepare_grade_sheet. The action field of this menu item is to send a message "prepare_grade_sheets of evaluation process". We extract the message id prepare_grade_sheet of the evaluation process. And lookup the STM for this message prepare_grade_sheet, and process as evaluation process. We find the message prepare_grade_sheet in STM. We then extract the following :-

- sender class DOAA_INTERFACE
- receiver class DOAA
- action field do_sub_process_prepare_grade_sheet

In case the message prepare_grade_sheet is not found in the STM, then there is an error in the specification. (This case has not been fully specified and hence is flagged as consistency error).

However the action here is to do a subprocess. So the subprocess id "prepare_grade_sheet" is extracted. All messages pertaining to this subprocess is stored

in STM, as a linked list. This linked list of messages constitute a part of the usecase. For each message, the message id is stored under the sender class and the receiver class. Within the class the messages are again stored as a linked list of messages. This is done because during testcase generation, the pre- and post-condition which restrict values of an attribute of classes are extracted from the messages of the usecase.

The message structure is stored under the sender class and the receiver class. The message structure is modified to point to the next message in the same usecase. Thus the message in the sender class points to the copy of the message stored under the receiver class. The receiver class message then points to the next message in the same usecase.

The action field of the message can then again be of the following type:-

- send message
- do subprocess (explained in example above)
- perform some action and stop
- there is no action field

If the action is to send a message, then again we extract the message id and the message is searched in the symbol table manager, message is stored under the sender class and receiver class and pointer to the next message in the same usecase is assigned. This sequence is performed until either the action field is null or action is to do some task and stop. Thus a sequence of messages for the usecase is stored under the respective classes.

If the action is to do a subprocess, then all the message sequences, stored under that subprocess in the symbol table manager, is taken and stored under respective classes and their pointers assigned accordingly. All messages of the usecase are connected by pointers, which point to the messages in the same usecase. Each usecase is given a unique usecase id.

When the action of the message is to perform a task and then stop, then this is taken as a termination point of the usecase. This is because the task will not be

stored in the symbol table manager as a message. Thus even if a search in the STM is carried out with the task as the message id, no message will be found. The last message in the usecase will have its next message in the usecase point to NULL.

Lastly when the action field of a message is missing then also its considered to be a termination of the usecase.

All the messages that are stored under a class have one more pointer that points to the next message under that class. The next message may or may not belong to the same usecase.

3.1.2 Do subprocess

When the action of the menu item selected involves activating a subprocess then all messages relating to the subprocess are stored in the STM under this subprocess id. The subprocess may contain a sequence of messages, which may be stored in a loop or in an if-else condition. We get the message sequence for the if condition as well as for the else condition. In case the message sequence is stored in the loop again all messages are extracted and stored under respective classes (as in the send message process), and appropriate pointers assigned. A subprocess would constitute a usecase or a part of usecase.

3.1.3 Invoke menu

When the action of the menu item selected is to invoke a submenu or another menu, then another pull down menu is provided to the user. The user may then select any menu item which would then point to the start of a usecase (provided the menu item selected is of the type send message or do subprocess or select fill in item. If the menu item again is of the type invoke menu then a menu is presented to the user once more).

3.1.4 Invoke fill in

When the action of the menu item selected is to invoke a fill in item, then the blank fields of the interface class are displayed to the user, where the user is required to

fill in some details. Fill in item are generally used to represent data in forms and to take some temporary input data. This is actually the interface through which the user is interacting with the system.

3.1.5 Select menu item

When the action of menu item is to select a menu item, the user selects a menu item from the menu displayed. Based on the menu item selected a usecase is generated or further a menu is presented or fill in interface is selected.

3.2 Testcases

Once a number of usecases for a particular interface class selected, are generated the same step is performed to get rest of the usecases for rest of the interface class. Each usecase will have a unique usecase id which identifies the useacase. Also all the messages for a particular class are stored under the class structure. A class will have a sequence of messages belonging to same or different usecase(s). All messages of a particular usecase would be linked accordingly. And all messages of a class would also be linked.

Thus now to generate testcases, we check a particular usecase and traverse it. Since usecase is stored as a sequence of messages, we keep checking the messages for precondition and postcondition. And based on these pre- and post- condition we generate testcases.

Precondition is the constraint on the state of the sender before sending a message and postcondition is the constraint on the state of the receiver before receiving the message. If a precondition is not satisfied then the sender will not send the message till the condition is satisfied. Similarly if a postcondition is not satisfied then the receiver will not receive the message till the condition is satisfied.

The pre- and post-condition is a combination (and/or) of one of the following:-

- received message id
- received resultof message id

- done subprocess
- constraints

The first three type of conditions help in determining the consistency in the requirement specification, whereas the fourth type of condition helps in determining the testcase. Each type of condition is dealt with separately.

3.2.1 Received message id

While traversing a usecase, we check the message for precondition and postcondition. In case the precondition is of the type received message id, then we extract the message id from the precondition. Also we extract the sender and the receiver class from the message structure. Now the sender (receiver in the postcondition) should have received this message before sending (receiving in postcondition) the message to the receiver (from the sender in postcondition). We then find the sender class (receiver class in postcondition) in the class base. Once the sender class is found, we traverse the messages stored under that class, during usecase generation. And try to find a message with the same message id as extracted from the precondition (postcondition). In case no match is found then this usecase is flagged as an error as the usecase may not be complete.

In case message with the message id as extracted from the precondition is found, the usecase id of the found message is extracted. And then the usecase with this usecase id is traversed, till the message id as extracted from the precondition is encountered. If upto this message id no other message with a precondition (or postcondition) is encountered then we proceed ahead in our original usecase, in which we had encountered the last precondition. However in the found usecase id if there is another message with a precondition then the whole process of testcases is repeated.

How then is the inconsistency in the requirement specification determined ? It is determined if in the found usecase , there is a precondition to receive a message, after the precondition in the original usecase. For example suppose we are traversing a usecase with usecase id 1. There is a message with precondition "received message

X" in this usecase. We extract the message X from the precondition of usecase id 1. Also we extract the sender class and the receiver class. We then find the sender class in the class base. Once the sender class is found in the class base we search for the message X under this class. Suppose there is a message X under this class which has a usecase id 5. We then traverse the usecase id 5 from the beginning till the message id with message X is encountered in this usecase. Also suppose before encountering the message X, there is a message in usecase id 5 with a precondition "received message Y". And this message appears in the usecase id 1 after the message with precondition "received message X". Then there is an inconsistency in the specification, which has to be corrected.

Similar procedure is adopted to find inconsistency arising out of the postcondition. Thus the "received message" type of pre/post-condition is used to find the inconsistency in the requirement specification.

3.2.2 Received resultof message

In this type of precondition the sender class is waiting for some response from the receiver class. The sender class must have asked the receiver class for some details. And before executing further, the sender should have received the response to its earlier message. Thus its concluded that the precondition is from the same usecase id.

Thus again we extract the sender class, receiver class and the message id from the precondition. We then search the message under the sender class and the same usecase id as the original one. The usecase is then traversed now to find the message with message id given in the precondition before the precondition is encountered.

In case the message is found, we proceed ahead with the last usecase being processed. In case the message is not found, then an error is flagged in that usecase. Thus detecting an inconsistency in the specification.

Similarly inconsistency may be detected by checking the postconditions.

For example in a usecase that we checked, class DOAA sends message give_rule to class RULE_BOOK. There is however a precondition received resultof_compute_cpi. We first extract the message id compute_cpi from this precondition. Then we extract

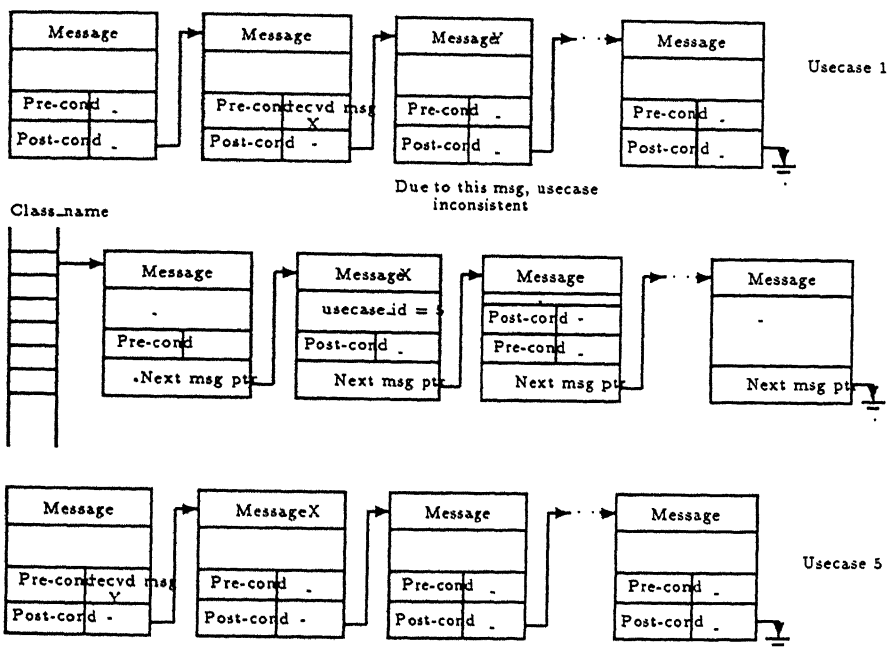


Figure 1: Checking for inconsistency

the sender class, i.e DOAA from the message. The class array is looked up for this class. And the pointer to the start of the message sequence stored under that class is taken. We then traverse, this message sequence of DOAA class till we get the message compute_cpi.

Once the message is found we take its usecase id. And now traverse this new found usecase from beginning to the message with message id compute_cpi. We keep checking every message for pre- and post- condition and thus try to determine consistency or inconsistency in the system as outlined in the above section (i.e. 3.2.1).

If the message is not found then there is an error in the specification.

3.2.3 Done subprocess

When the precondition encountered is of the type, done subprocess, then we conclude that this sender class should have received the message to do this subprocess earlier. Thus we extract the process id from the precondition and the sender class name. Then the messages under that sender class is checked in which this class is the receiver of the message "do subprocess" and the action field related to this message is of the type "do subprocess". If no such message is found, then flag an error in the usecase. Else if usecase is same as that of the present usecase then proceed as in 3.2.2. If the usecase is different than the present usecase then proceed as in 3.2.1 . Similarly we proceed in the case of postcondition. Thus inconsistency in the specification is determined.

For example in a usecase that we tested for consistency, class DOAA sends message stud_grade_sheet to class DEPTT. The precondition is done sub_process prep_grade_sheet. We first extract the process id prep_grade_sheet from this precondition. Also we take the sender class, i.e. DOAA. And now lookup the class array for class DOAA. We get the pointer to message sequence stored under this class and traverse it. We look for a message in which the receiver of the message is this class (i.e DOAA class) and the action of the message is to do subprocess prep_grade_sheet.

If the message is found then we proceed with the previous usecase. If the message is not found then then the previous usecase is flagged as INCORRECT.

3.2.4 Constraints

In case the precondition is of the type constraint, then the various constraints are flashed to the user who then gives the input which satisfies all these constraints. Here the input from the user is treated as string and the data input by the user may or may not satisfy the constraint. In either case, the data input by the user is recorded as a testcase. The user will be prompted as to whether the constraints can be satisfied or not. In any case if the user inputs data whether correct or incorrect it will be stored as a testcase.

Only the last type of precondition is actually used to generate the testcase directly from the requirement specification.

For example if there is a constraint that a student will be issued a warning letter if the SPI or the CPI at the end of the regular semester is

2.0 < SPI <= 4.5 AND CPI > 4.5

This constraint will be displayed to the user. And if there is any student who satisfies the above condition of SPI and CPI then the DOAA will issue message warning to the student. However, the user is displayed only the constraints. The data filled by the user is treated as a string and stored as testcase. If there is a value which would satisfy the above constraints, then it is stored as correct testcase. Values not satisfying the constraint are stored as incorrect testcase.

Chapter 4

Implementation

In this chapter is described, the various structures, used and modified, to implement the testcase generator. But the ones that were modified are only described here. The structures have been defined and declared in the thesis being done currently.

4.1 Use_Case_Msg

This structure uses the basic structure of the message, as described in the "proc.h" header file. The structure is as follows:-

struct Use_Case_Msg Msg u_c_msg; int use_case_id; Use_Case_Msg *next; Use_Case_Msg *nextmsg; ;

The variable u_c_msg is of the type Msg, the basic structure of message. An integer use_case_id defines the message belonging to a particular usecase uniquely. Pointer to data of the same type, points to the next message in the same usecase only. Also a pointer to the same data type again points to the next message in the same class. The testcase generator uses this structure to generate testcases and usecases.

4.2 newClass

This structure is used to store all the classes appearing in the requirement specification. All classes are stored in an array of this new class structure. The newClass structure is as follows:-

struct newClass Clas nclas; Use_Case_Msg *next; cls_array[];

The variable nclas is of the type Clas described in the "class.h" header file. As the individual class will have to point to sequences of messages a pointer to the data type Use_Case_Msg is also defined. Access to the messages associated under a class can be gained through this Use_Case_Msg pointer. This structure is initialized and filled during usecase generation, but is used extensively in testcase generation.

4.3 Start_ptr_table

All the usecases are stored in an array of structures called the start_ptr_table. The structure used to describe this table is as follows:-

struct Stat_Ptr_Table Use_Case_Msg *start_ptr; Use_Case_msg *curr_msg_ptr; int status_flag; start_ptr_table;

This table has a pointer to the first message in the usecase. Thus there is a pointer to data type Use_Case_Msg. The second pointer to the same data type keeps track of the message in the usecase upto which the usecase has been checked. The integer type of data to maintain the status of the usecase is used as follows:-

0 usecase is uninitialized 1 usecase has been checked and is found correct 2 usecase has been checked and is incorrect 3 usecase is being checked

4.4 Implementation

Initially the class array is initialized to null and the pointer to message is assigned to NULL. Then the start pointer table is initialized, wherein the curr_msg_ptr is assigned NULL and the status_flag of all the usecase is set to 0. Then each interface class is taken and usecases are generated. Finally considering each usecase either inconsistency in requirement specification is reported or testcase is generated.

Chapter 5

CONCLUSION

5.1 Conclusion

The complete implementation of the testcase generator was done in two steps. First, usecase generation module was tried. The examples taken were from DOAA office automation. As the complete specification would have been quite large, only registration and evaluation process were taken as example. An interface class was taken and usecases generated, for the various menu items in this interface class. This process of usecase generation was then incorporated for other interface classes.

Secondly, testcase generation module was developed. Aim of this module was to determine the inconsistencies and to present the testcases to the user. The user would then have to store his input as testcases for testing the system. The user in this case is the developer of the system who can store correct or incorrect data as testcases to check for the correctness of the system being built.

5.2 Results

Usecases were generated for all the interface classes in DOAA office automation system (i.e registration and evaluation process). Only then testcases were generated and testing was done. The example taken lends itself for testing, i.e check for inconsistencies in the specification. Whenever at a particular message, the usecase

was found to be incorrect, the testing was not stopped. It was continued till the end of the usecase, which helps in determining the place where that usecase has failed.

For generating the testcases separate example of academic probation was taken. When this tool was used on the academic probation process testcases were generated and prompted to the user. If there was a condition that satisfied the constraint, it was taken as a correct testcase. Values not satisfying the constraint were taken as incorrect testcases.

5.3 Possible extensions

1. This tool can be used to generate testcases (both correct and incorrect) automatically.

2. Can also be used to validate the system being built, by running testcases being generated on the software being done.

Appendix A

Pseudocodes

A.1 Pseudocode for usecase generation

```
traverse_menu_tree(curr_menu_ptr)
{
 if(curr_menu_ptr -> action = invoke_menu)
   traverse_menu_tree(curr_menu_ptr -> down);
 else
 {
   if(curr_menu_ptr -> action = send_msg)
   {
    generate_usecase_id;
    scan_the_sequence_of_messgae;
   }
   else
   {
     if(curr_menu_ptr -> action = do_sub_process)
     {
      generate_usecase_id;
      get_all_messages_related_to+sub_process;
```

```
    }
    else
     give_user_interface_to_user;
   }
 }
 if(curr_menu_ptr -> next != NULL )
   traverse_menu_tree(curr_menu_ptr -> next)
}


// scans the sequence of message

scan_the_sequence_of_message(curr_menu_ptr)
{
  while (curr_menu_ptr -> action == send_msg)
  {
   get_message_id(curr_menu_ptr);
   initialize_this_message_to_new_message_structure(msd_id);
   associate_this_message_to_appropriate_class(msg_id);
  }
}


// gets and associates all messages related to the subprocess

get_all_messages_related_to_sub_process(curr_menu_ptr)
{
  while(no_more_messages_in_sub_process)
  {
   get_next_msg_of_this_sub_process;
   initialize_this_message_to_new_message_structure(msd_id);
```

```
   associate_this_message_to_appropriate_class(msg_id);
  }
}
```

A.2 Pseudocode for testing and testcase generation

```
for_each_usecase do
{
  traverse_usecase();
  {
    while(starting_message != message_being_checked)
    {
      switch(message -> precondiion)
      {
         case RECD_MSG :
           search_sender_class_name_in_class_array;
           get_ptr_to_msg_seq_for_this_class;
           traverse_this_msg_seq_for_msg_id_as_in_RECD_MSG;
           if(message_not_found)
           {
            flag_usecase_as_INCORRECT;
            break;
           }
           else
           {
            check_usecase_id_of_this_found_msg;
            check_status_flag_of_that_usecase;
```

```
switch(status_flag)
{

   case CORRECT :
     usecaseIis_correct;
     break;

   case INCORRECT :
     flag_usecase_as_INCORRECT:
     break;

   case BEING_CHECKED :
     check_for_found_msg_in_this_usecase;
     if(found)
       break;
     else
       flag_usecase_as_INCORRECT;
     break;

    case UNINITIALIZED :
      traverse_usecase();
      break;
  }
  break:
 }

case RECD_RESULTOF_MSG :
  search_sender_class_name_in_class_array;
  get_ptr_to_msg_seq_for_this_class;
  extract_msd_id_from_RECD_RESULTOF_MSG;
  traverse_this_msg_seq_for_msg_id_as_in_RECD_RESULTOF_MSG;
```

```
  if(message_not_found)
  {
   flag_usecase_as_INCORRECT;
   break;
  }
  else
  {
    check_usecase_id_of_this_found_msg;
    check_found_msg_in_usecase;
    if(found)
      break;
    else
    {
      flag_usecase_as_INCORRECT;
      break;
    }
  }
  break;

case DONE_SUB_PROCESS :
  search_sender_class_name_in_class_array;
  get_ptr_to_msg_seq_for_this_class;
  traverse_this_msg_seq_for_msg_id_where_class_is_RECEIVER;
  if(message_not_found)
  {
    flag_usecase_as_INCORRECT;
    break;
  }
  break;

case CONSTRAINT :
```

```
      display_constraint_to_the_user;
      take_input_from_the_user_and_store_as_testcase;
      break;
}

switch(message -> postcondiion)
{
   case RECD_MSG :
   {
      search_receiver_class_name_in_class_array;
      get_ptr_to_msg_seq_for_this_class;
      traverse_this_msg_seq_for_msg_id_as_in_RECD_MSG;
      if(message_not_found)
      {
       flag_usecase_as_INCORRECT;
       break;
      }
      else
      {
        check_usecase_id_of_this_found_msg;
        check_status_flag_of_that_usecase;

        switch(status_flag)
        {
           case CORRECT :
             usecaseIis_correct;
             break;

           case INCORRECT :
             flag_usecase_as_INCORRECT:
             break;
```

```
        case BEING_CHECKED :
          check_for_found_msg_in_this_usecase;
          if(found)
            break;
          else
            flag_usecase_as_INCORRECT;
          break;

          case UNINITIALIZED :
            traverse_usecase();
            break;
    }
  }
  break:
}

case RECD_RESULTOF_MSG :
{
  search_receiver_class_name_in_class_array;
  get_ptr_to_msg_seq_for_this_class;
  extract_msd_id_from_RECD_RESULTOF_MSG;
  traverse_this_msg_seq_for_msg_id_as_in_RECD_RESULTOF_MSG;
  if(message_not_found)
  {
    flag_usecase_as_INCORRECT;
    break;
  }
  else
  {
    check_usecase_id_of_this_found_msg;
```

```
    check_found_msg_in_usecase;
    if(found)
      break;
    else
    {
      flag_usecase_as_INCORRECT;
      break;
    }
   }
   break;
}

case DONE_SUB_PROCESS :
{
    search_receiver_class_name_in_class_array;
    get_ptr_to_msg_seq_for_this_class;
    traverse_this_msg_seq_for_msg_id_where_class_is_RECEIVER;
    if(message_not_found)
    {
      flag_usecase_as_INCORRECT;
      break;
    }
    break;
   }

    case CONSTRAINT :
    {
      display_constraint_to_the_user;
      take_input_from_the_user_and_store_as_testcase;
      break;
    }
```

```
        }
        starting_message = starting_message -> next;
      }
    }
  }
```

Appendix B

Example of DOAA office automation

```
#-----------------------------------------------------------------

CLASS : PERSON

TYPE : ABSTRACT

INHERITS : PERSON

GENERALISATION OF : STUDENT, FACULTY_MEMBER

ATTR : name <STRING>,
       email_addr <STRING>,
       address <STRING>

#-----------------------------------------------------------------

CLASS : STUDENT
```

```
TYPE : ABSTRACT

INHERITS : PERSON

GENERALISATION OF : UG_STUDENT, PG_STUDENT

ATTR : roll_no <NUMBER>,
       address <STRING>,
       joining_date   <DATE>, #date of joining the institute
       semester  < NUMBER > # Academic semester for student

RELATED WITH : COURSE (takes, N),
               PERFORMANCE_RECORD (has, 1)

#-----------------------------------------------------------------

CLASS : UG_STUDENT

INHERITS : STUDENT

ATTR : acad_status < STRING > # regular/warning/probation  etc

RELATED WITH : DUGC (CONVENER, 1)

#-----------------------------------------------------------------

CLASS : PG_STUDENT

TYPE : ABSTRACT

INHERITS : STUDENT
```

```
GENERALISATION OF : MTECH_STUDENT, PHD_STUDENT

ATTR : leave_status <{ casual_leaves <NUMBER>, sick_leaves <NUMBER>,
                       vacation_leaves <NUMBER> }>

-----------------------------------------------------------------

CLASS : MTECH_STUDENT

INHERITS : PG_STUDENT

RELATED WITH : DPGC (CONVENER, 1),
               THESIS_SUPERVISOR (Th_guide, 1),
               TA_SIPERVISOR (Ta_instr, 1)

#----------------------------------------------------------------

CLASS : PHD_STUDENT

INHERITS : PG_STUDENT

RELATED WITH : DPGC (CONVENER, 1),
               THESIS_SUPERVISOR (Th_guide, 1),
               PHD_ACAD_DETAILS (acad_details, 1)

#----------------------------------------------------------------

CLASS : PHD_ACAD_DETAILS

ATTR : seminars_given <TABLE [ type <STRING>, title <STRING>, date <DATE> ]>
```

```
#-----------------------------------------------------------------

CLASS : PERFORMANCE_RECORD

ATTR : semester <NUMBER>,
       year <NUMBER>,
       cpi< NUMBER>,
       spi< NUMBER>,
       text<STRING>,
       grades < TABLE [ c_no<NUMBER>,units<NUMBER>,grade<NUMBER>] >

#-----------------------------------------------------------------

CLASS : REG_FORM

ATTR : year< NUMBER>,
       sem< NUMBER>,
       roll_no< NUMBER>,
       room_no  < STRING >,
       student_name<STRING>,
       Thesis_superviser<STRING>,
       Financial_Asst <STRING>,
       DPGC_sign<STRING>,
       DUGC_sign  <STRING>,
       courses < TABLE [ course_no <NUMBER>, c_name <STRING>,
                         units <NUMBER>, instructor <STRING> ] >

#-----------------------------------------------------------------

CLASS : FACULTY_MEMBER
```

```
INHERITS : PERSON

GENERALISATION OF : COURSE_COORDINATOR, CONVENER, DOAA

ATTR : pf_no <NUMBER>,
       leave_status <{ casual_leaves <NUMBER>, sick_leaves <NUMBER>,
                       vacation_leaves <NUMBER>}>

#-----------------------------------------------------------------------#

CLASS : COURSE

ATTR :
       c_no <NUMBER>,
       c_name <STRING>,
       units_pg <NUMBER>,
       units_ug  <NUMBER>,
       syllabus <STRING>,
       references  <STRING>,
       approval_date  <DATE>,
       lab <NUMBER>,
       lecture <NUMBER>,
       tutorial  <NUMBER>,
       course_slots < TABLE [ time_slot<NUMBER>,day<STRING>,room_no<STRING>]

RELATED WITH : COURSE_COORDINATOR (taught_by, N),
               STUDENT (enrolled_by, N)

#-----------------------------------------------------------------
```

CLASS : COURSE_COORDINATOR

INHERITS : FACULTY_MEMBER

RELATED WITH : COURSE (teaches, N)

#----------

CLASS : CONVENER

TYPE : ABSTRACT

GENERALISATION OF : DPGC, DUGC

RELATED WITH : RULE_BOOK (Rules_store, 1)

#----------

CLASS : DPGC

INHERITS : CONVENER

#----------

CLASS : DUGC

INHERITS : CONVENER

#----------

CLASS : DOAA

```
INHERITS : FACULTY_MEMBER

RELATED WITH : DPGC (DPGC, 1),
               DUGC (DUGC, 1),
               RULE_BOOK (Rules_store, 1)

#-----------------------------------------------------------------

CLASS : NOTICE_BOARD

AGGREGATION OF : NOTICE (N)

#-----------------------------------------------------------------

CLASS : NOTICE

ATTR : date <DATE>,
       issued_by <STRING>,
       title <STRING>,
       text < STRING>

#-----------------------------------------------------------------

CLASS : RULE_BOOK

RELATED WITH : RULE (has, N)

#-----------------------------------------------------------------

CLASS : RULE
```

```
ATTR : rule_id <STRING>,
       w_e_f <DATE>,
       eff_upto <DATE>,
       rule_func <STRING>

#-----------------------------------------------------------------

CLASS : TIME_TABLE

ATTR : time_table < TABLE [ c_no <NUMBER>, Instructor <STRING>, schedule
                          <TABLE [ day <STRING>, time_slot <NUMBER>] > ] >

RELATED WITH              : DEPARTMENT ( Belongs_to , 1)

#-----------------------------------------------------------------

CLASS                     : DEGREE

ATTR                      : degree_name <STRING>,
                            description <STRING>

RELATED WITH              : DEPARTMENT (OfferedBy, M-N),
                            CERTIFICATE (certificate, 1)

#-----------------------------------------------------------------

CLASS                     : LEAVE_FORM

GENERALISATION OF         : STDENT_LEAVE_FORM, FACULTY_LEAVE_FORM
```

```
ATTR                    :
                from <DATE>,
                to <DATE>,
                type_of_leave <STRING>

#-----------------------------------------------------------------

CLASS                   : STUDENT_LEAVE_FORM

INHERITS                : LEAVE_FORM

ATTR                    :
                roll_no_of_stud <NUMBER>,
                name_of_stud <STRING>,
                stud_sign <STRING>,
                ta_supervisor_sign <STRING>,
                thesis_supervisor_sign <STRING>

#-----------------------------------------------------------------

CLASS                   : FACULTY_LEAVE_FORM

INHERITS                : LEAVE_FORM

ATTR                    :
                pf_no <NUMBER>,
                name <STRING>,
                sign <STRING>

#-----------------------------------------------------------------
```

```
CLASS                          : DEPARTMENT

RELATED WITH                   :
        STUDENT ( Has , 1-N) ,
        FACULTY ( Has , M-N) ,
        NOTICE_BOARD ( Has , 1-1) ,
        TIME_TABLE (Has, 1-1)

#-----------------------------------------------------------------

CLASS : ACCOUNTS_SECTION

INHERITS : GENERAL_ADMINISTRATOR

#-----------------------------------------------------------------

CLASS : HALL_OFFICE

INHERITS : GENERAL_ADMINISTRATOR

#-----------------------------------------------------------------

CLASS : CALENDER

ATTR : calender <TABLE [ date <DATE>, event <STRING>,
                         semester <STRING>] >,
       current_sem <NUMBER>,
       current_year <NUMBER>

#-----------------------------------------------------------------
```

```
CLASS    : PERSON_INTERFACE

TYPE     : INTERFACE

MENU ITEMS    :
            { ID : "calender"
      NAME : "See Calender"
      ACTION : SEND MESSAGE display_calender OF queries
    },
    { ID : "notice"
      NAME : "See Notice Board"
      ACTION : SEND MESSAGE display_notices OF queries
    },
    { ID : "student_details"
      NAME : "Show details"
      ACTION : SEND MESSAGE give_details OF queries
      DESCRIPTION : "Shows the given student's details that are not
     restricted by the student"
    }

#-----------------------------------------------------------------

CLASS                        : STUDENT_INTERFACE

TYPE                         : INTERFACE

INHERITS     : PERSON_INTERFACE

MENU ITEMS                   :
             { ID : "register"
               NAME : "Registration"
```

```
  ACTION : INVOKE_MENU [ "register/pay_hall_dues",
                         "register/pay_instt_dues",
                         "register/get_reg_form",
                         "register/fill_reg_form",
                         "register/send_reg_form"
                       ]
},
{ ID : "leaves"
  NAME : "Leaves"
  ACTION : INVOKE_MENU [ "leaves/get_leave_form",
                         "leaves/fill_leave_form",
                         "leaves/send_leave_form"
                       ]
},
{ ID : "add_drop"
  NAME : "Add / Drop Courses"
  ACTION : INVOKE_MENU [ "add_drop/get_add_drop_form",
                         "add_drop/fill_add_drop_form",
                         "add_drop/submit_add_drop_form"]
},
{ ID : "register/pay_hall_dues"
  NAME : "pay Hall dues"
  ACTION : SEND MESSAGE pay_hall_dues OF Registration
},
{ ID : "register/pay_instt_dues"
  NAME : "pay Institute dues"
  ACTION : SEND MESSAGE pay_instt_dues OF Registration
},
{ ID : "register/get_reg_form"
  NAME : "Get Regestration Form"
  ACTION : SEND MESSAGE get_reg_form OF Regestration_process
```

```
},
{ ID : "register/fill_reg_form"
  NAME : "Fill Regestration Form"
  ACTION : SEND MESSAGE display_method_fill_reg_form OF
           Regestration_process
},
{ ID : "register/send_reg_form"
  NAME : "Submit Regestration Form"
  ACTION : SEND MESSAGE submit_filled_reg_form OF
           Regestration_process
},
{ ID : "add_drop/get_add_drop_form"
  NAME : "Get Add Drop Form"
  ACTION : SEND MESSAGE get_add_drop_form OF Add_Drop_process
},
{ ID : "add_drop/fill_add_drop_form"
  NAME : "Fill Add/Drop Form"
  ACTION : SEND MESSAGE display_method_fill_add_drop_form OF
           fill_add_drop_form_process
},
{ ID : "add_drop/submit_add_drop_form"
  NAME : "Submit Add Drop Form"
  ACTION : SEND MESSAGE send_add_drop_form OF add_drop_process
},
{ ID : "leaves/get_leave_form"
  NAME : "Get Leave Form"
  ACTION : SEND MESSAGE give_leave_form OF Leave_processing
},
{ ID : "leaves/fill_leave_form"
  NAME : "Fill the Leave Form"
  ACTION : SEND MESSAGE fill_leave_form OF Leave_processing
```

```
            }

DISPLAY METHODS :

        MENU DISPLAY : VERTICAL

RELATED WITH     : A_D_FORM_INTERFACE (uses, 1-1),
                   STUDENT (attached_to, 1-1)

#-----------------------------------------------------------------

CLASS   : REG_FORM_INTERFACE

TYPE    : INTERFACE

MENU ITEMS :
                 { ID : "store"
                   NAME : "store this reg form"
                   ACTION : SEND MESSAGE store_reg_form
                 }

 FILLIN ITEMS :
    { ID : "year"
    NAME : "Year"
    CONSTR : TYPE = < NUMBER>
  },
    { ID : "sem"
    NAME : "Semester"
    CONSTR :  TYPE = <NUMBER>
  },
    { ID : "roll_no"
```

```
  NAME : "Roll Number"
  CONSTR : TYPE =  < NUMBER>
},
  { ID : "room_no"
  NAME : "Room Number"
  CONSTR : TYPE = < STRING >
},
  { ID : "student_NAME"
  NAME : "Name in Block letters"
  CONSTR : TYPE =  <STRING>
},
{ ID : "fin_asst"
  NAME : "Financial_Asst"
  CONSTR : TYPE = <STRING>
},
  { ID : "th_sup"
  NAME : "Thesis_superviser"
  CONSTR : TYPE = <STRING>
},
{ ID : "convener_sign"
  NAME : "Signature OF the Convener (DPGC / DUGC)"
  CONSTR : TYPE = <STRING>
},
{ ID : "course_no"
  NAME : "Course Number"
  CONSTR : TYPE = <STRING>
},
{ ID : "c_NAME"
  NAME : "Course Name"
  CONSTR : TYPE = <STRING>
},
```

```
{ ID : "units"
  NAME : "No OF Units"
  CONSTR : TYPE = <NUMBER>
},
{ ID : "instr"
  NAME : "Initials OF Instructor"
  CONSTR : TYPE = <STRING>
},
  { ID : "courses"
  NAME : "Courses to register"
  CONSTR : TYPE = < TABLE [ course_no<NUMBER>,
  c_NAME<STRING>, units<NUMBER>,
    instructor<STRING> ] >
          }

DISPLAY METHODS :

        FILLIN DISPLAY {        # This is to fill the Regestration form
                "Regestration Form"

                <FILLIN ITEMS : year, sem>
                <FILLIN ITEMS : name>
                <FILLIN ITEMS : roll_no>
                <FILLIN ITEMS : room_no>
                <FILLIN ITEMS : courses >

         }

#-----------------------------------------------------------------

CLASS   :     DOAA_INTERFACE
```

```
TYPE     :        INTERFACE

INHERITS :  FACULTY_MEMBER_INTERFACE

MENU ITEMS : { ID : "regestration"
               NAME : "Regestration related"
               ACTION : INVOKE_MENU [ "regestration/send_reg_forms" ]
             },
             { ID : "grades"
               NAME : "Grades processing"
               ACTION : INVOKE_MENU [ "grades/prepare_grade_sheets",
                                      "grades/send_grade_sheets_to_depts" ]
             },
     { ID : "degree_award"
       NAME : "degree Awarding"
       ACTION : INVOKE_MENU [ "degree_award/check_for_deg_prereq" ]
     },
             { ID : "regestration/send_reg_forms"
               NAME : "Send Regestration forms to DPGCs, DUGCs"
               ACTION : SEND MESSAGE send_reg_forms_to_cdgcs OF
                        Regestration_process
             },
             { ID : "grades/prepare_grade_sheets"
               NAME : "Prepare Student's Grade Sheets"
               ACTION : SEND MESSAGE prepare_grade_sheets OF Evaluation_process
             },
             { ID : "grades/send_grade_sheets_to_depts"
               NAME : "Send Student's Grade Sheets to their Depts"
               ACTION : SEND MESSAGE students_grade_sheets OF
                        Evaluation_process
```

```
            },
     { ID : "send_notice"
       NAME : "Send a Notice"
       ACTION : DO SUB_PROCESS display_method
     }

#-------------------------------------------------------------------------

CLASS            : COURSE_COORD_INTERFACE

TYPE             : INTERFACE

MENU ITEMS       :
                   { ID : "courses"
                     NAME : "Courses Menu"
                     ACTION : INVOKE_MENU ["general/get_registered_students"]
                   },
                   { ID : "courses/give_grades"
                     NAME : "Give grades to registered students"
                     ACTION : DO SUB_PROCESS take_input_course_number;
                              SEND MESSAGE give_registered_students OF
                                   Evaluation ;
                              DO SUB_PROCESS display_method_give_grades
                   }

FILLIN ITEMS     : { ID : "student"
     NAME : "Student Name"
     CONSTR : TYPE = <STRING>
   },
   { ID : "roll_no"
     NAME : "Roll number"
```

```
    CONSTR : TYPE = <NUMBER>
  },
  { ID : "grade"
    NAME : "Grade"
    CONSTR : TYPE = <CHAR> && VALUE = [A-F]
  },
  { ID : "grade_list"
    NAME : "List OF Student Grades"
    CONSTR : TYPE = < TABLE [student<STRING>, roll_no
<NUMBER>, grade_given<CHAR>]>
                  }

DISPLAY METHODS :
MENU DISPLAY : VERTICAL
#-------------------------------------------------------------------------#

CLASS : NOTICE_INTERFACE
TYPE  : INTERFACE

FILLIN ITEMS : { ID : "title"
    NAME : "Title OF the Notice"
    CONSTR : TYPE = <STRING>
                    ATTACHED_TO : NOTICE
  },
  { ID : "to"
    NAME : "To"
    CONSTR : TYPE = <STRING>
                    ATTACHED_TO : NOTICE
  },
                  { ID : "timestamp"
                    NAME : "Dated"
```

```
                CONSTR : TYPE = <DATE>
                ATTACHED_TO : NOTICE
              },
              { ID : "issuedby"
                NAME : "Issued by"
                CONSTR : TYPE = <STRING>
                ATTACHED_TO : NOTICE
              },
              { ID : "text"
                NAME : "Text OF the Notice"
                CONSTR : TYPE = <STRING>
                ATTACHED_TO : NOTICE
              }

RELATED WITH    : NOTICE (notice, 1)

#-------------------------------------------------------------------------
GENERIC PROCESS :: make_notice
INPUTS : ISSUEE
#--------------------------------#

FROM       : ISSUE
TO         : NOTICE_INTERFACE
MSG_ID     : make_notice ()
RESULT     : res_notice <NOTICE>

FROM       : NOTICE_INTERFACE
TO         : ISSUE
MSG_ID     : replyto_make_notice (notice <NOTICE>)
MSG_TYPE   : reply
```

```
MSG_ID     : extract_details (query = "roll_no" <STRING>, reg_form
                              <REGESTRATION_FORM>)
             # extract_details should be taken as a general purpose message
RESULT     : roll_no <NUMBER>

FROM       : DOAA
TO         : HALL_OFFICE
MSG_ID     : check_student_for_dues (roll_no <NUMBER>)
RESULT     : payed <BOOLEAN>

IF (payed == TRUE) {

    FROM       : DOAA
    TO         : ACCOUNT_SECTION
    MSG_ID     : check_student_for_dues (roll_no <NUMBER>)
    RESULT     : payed_instt_dues <BOOLEAN>

    IF (payed_instt_dues == TRUE) {

        FROM       : DOAA
        TO         : DOAA
        MSG_ID     : form_links ()
        ACTION     : DO SUB_PROCESS form_assoc_between_stud_course
                                    ( reg_form <REGESTRATION_FORM>)

    } ELSE {

        FROM       : DOAA
        TO         : STUDENT (BACK)
        MSG_ID     : replyto_req_for_reg (remark = "Instt dues not paid"
                                          <STRING>)
```

```
        MSG_TYPE   : reply

    }

} ELSE {

    FROM       : DOAA
    TO         : STUDENT (BACK)
    MSG_ID     : replyto_req_for_reg (remark = "Hall duejs not paid" <STRING>)
    MSG_TYPE   : reply

}

#-------------------------------------------------------------------------#

SUB_PROCESS :: form_association_between_student_course
#-----------------------------------------------------#

FROM       : DOAA
TO         : DOAA
MSG_ID     : extract_details (query = "roll_no, courses" <STRING>,
                              reg_form <REGESTRATION_FORM>)
RESULT     : res {roll_no <NUMBER>, courses <TABLE [c_no <STRING>, c_name <STRING
                                        units <NUMBER>, instructor <STRING>]>}

FOREACH c_no (courses) {

    FROM       : DOAA
    TO         : COURSE (c_no == COURSE::c_no)
    MSG_ID     : give_handle ()
    MSG_TYPE   : shared
```

```
    RESULT     : course_ref <HANDLE>

    FROM       : DOAA
    TO         : STUDENT (roll_no == STUDENT::roll_no)
    MSG_ID     : give_handle ()
    MSG_TYPE   : shared # for security
    RESULT     : student_ref <HANDLE>

    # Here comes the necessity for the design to have an idea about the
    # implementation platform
    #.For the above cause (of forming association between students and
    # courses), if we are ultimately going to build the application over
    # C++, above specification will be OK But if we are going to build on
    # ORACLE, then it may not look proper as in oracle there is no concept
    # of references

    FROM       : DOAA
    TO         : STUDENT (roll_no == STUDENT::roll_no)
    MSG_ID     : form_assoc_with_course (course_ref <REFERENCE>)
    MSG_TYPE   : exclusive

    FROM       : DOAA
    TO         : COURSE (c_no == COURSE::c_no)
    MSG_ID     : form_assoc_with_student (student_ref <REFERENCE>)
    MSG_TYPE   : exclusive

}

#-------------------------------------------------------------------------#
```

```
SUB_PROCESS :: change_the_reg_form
#--------------------------------#

 FROM                  : STUDENT
 TO                    : STUDENT
 MSG_ID                : retrive_reg_form ( constraints <STRING> )
 RESULT                : reg_form <REG_FORM>
 ACTION                : SEND MESSAGE modify_attr


 FROM                  : STUDENT
 TO                    : STUDENT
 MSG_ID                : modify_attr ( reg_form <REG_FORM>,
                                       values_string <STRING> )

#-------------------------------------------------------------------#

SUB_PROCESS :: check_and_sign_reg_form
#-------------------------------------#

FROM       : CONVENER
TO         : CONVENER
MSG_ID     : check_reg_form (reg_form <REGESTRATION_FORM>)
RESULT     : correct <BOOLEAN>

IF (correct == FALSE) {

    FROM       : CONVENER
    TO         : STUDENT (BACK)
    MSG_ID     : replyto_sign_reg_form (remarks = "Incorrect" <STRING>)
    MSG_TYPE   : reply
```

```
} ELSE {

    FROM       : CONVENER
    TO         : REGESTRATION_FORM
    MSG_ID     : put_dpgc_sign (signature <SIGNATURE>)
    MSG_TYPE   : exclusive

}

#-------------------------------------------------------------------------------#


SUB_PROCESS :: check_eligibility_for_reg
#--------------------------------------#

FROM       : CONVENER
TO         : CONVENER
MSG_ID     : check_for_reg_notice ()
RESULT     : present <BOOLEAN>

IF (RECEIVED reg_notice OF Regestration) {

    FROM       : CONVENER
    TO         : STUDENT (BACK)
    MSG_ID     : give_details (str = "status,desgn" <STRING>)
    RESULT     : details <STRING>

    FROM       : STUDENT
    TO         : CONVENER (BACK)
    MSG_ID     : replyto_give_details (details <STRING>)
```

```
FROM       : CONVENER
TO         : RULE_BOOK
MSG_ID     : give_rule (str="reg_elig_check_rule" <STRING>, desgn <STRING>)
RESULT     : rule <RULE>

FROM       : CONVENER
TO         : CONVENER
MSG_ID     : apply_rule (rule <RULE>, details <STRING>)
RESULT     : eligibile <BOOLEAN>

IF (eligible == TRUE) {

    FROM       : CONVENER
    TO         : CONVENER
    MSG_ID     : create_reg_form ()
    RESULT     : reg_form <REGESTRATION_FORM>

    FROM       : CONVENER
    TO         : STUDENT (BACK)
    MSG_ID     : replyto_req_for_reg_form ( reg_form <REGESTRATION_FORM>,
                        remarks = "Eligible for Regestration" <STRING> )
    MSG_TYPE   : reply
    ACTION     : DO SUB_PROCESS store (reg_form <REGESTRATION_FORM>),
                 DO SUB_PROCESS store_temp (remarks <STRING>)

} ELSE {  # eligible is FALSE

    FROM       : CONVENER
    TO         : STUDENT (BACK)
    MSG_ID     : replyto_req_for_reg_form (reg_form <REGESTRATION_FORM>,
```

```
                    remarks = "Not eligible for regestration" <STRING>)
                # reg_form will be NULL

        MSG_TYPE  : reply
        ACTION    : DO SUB_PROCESS store (reg_form <REGESTRATION_FORM>),
                    DO SUB_PROCESS store_temp (remarks <STRING>)

    }

} ELSE { # Regestration notice has not been received from DOAA
         # means it is not yet time for regestration

    FROM       : CONVENER
    TO         : STUDENT (BACK)
    MSG_ID     : replyto_req_for_reg_form (reg_form <REGESTRATION_FORM>,
                        remarks = "not time for regestration" <STRING>)
    MSG_TYPE   : reply
    ACTION     : DO SUB_PROCESS store (reg_form <REGESTRATION_FORM>),
                 DO SUB_PROCESS store_temp (remarks <STRING>)

}

#-------------------------------------------------------------------------#

#-----------------------------------------------------------

SUB_PROCESS :: prepare_studs_grade_sheets
#-------------------------------------#
# check_whether_all_grades_received checks whether DOAA has received grades
# from all course coordinators
```

```
FROM       : DOAA
TO         : DOAA
MSG_ID     : check_whether_all_grades_received ( )
RESULT     : res { all_courses_received <BOOLEAN>, course_grades_not_received
                                          <TABLE[c_no <NUMBER>]> }

IF (all_courses_received == TRUE) {

        FROM       : DOAA
        TO         : RULE_BOOK
        MSG_ID     : give_rule ( query_str <STRING> )
        MSG_COND   : (all_courses_received == TRUE)
        RESULT     : cpi_compute_rule <RULE>

        FROM       : DOAA
        TO         : DOAA
        MSG_ID     : retreive_grade_lists()
        RESULT     : stud_grade_list < TABLE [ stud_name <STRING>,
                     roll_no <NUMBER>, grade <CHAR>]>

        FROM       : DOAA
        TO         : DOAA
        MSG_ID     : create_stud_grade_sheets (stud_grade_list < TABLE
                     [ stud_name <STRING>, roll_no <NUMBER>,
                     grade <CHAR>]> )
        RESULT     : stud_grade_sheet <TABLE [c_no <STRING>,
                      c_name<STRING>, units <NUMBER>, grade <CHARACTER>]>
        PRE_COND   : RECEIVED RESULTOF retreive_grade_lists

        FROM       : DOAA
```

```
TO          : DOAA
MSG_ID      : compute_cpi ( cpi_compute_rule <RULE>,
              stud_grade_sheet <TABLE [c_no <STRING>,
               c_name <STRING>, units <NUMBER>, grade <CHARACTER>]>)
RESULT      : res {stud_grade_sheet <TABLE [c_no <STRING>,
               c_name <STRING>, units <NUMBER>,
               grade <CHARACTER> ] >, stud_cpi <NUMBER> }
PRE_COND    : RECEIVED RESULTOF create_stud_grade_sheets

FROM        : DOAA
TO          : RULE_BOOK
MSG_ID      : give_rule ( query_str <STRING> )
RESULT      : status_eval_rule <RULE>
PRE_COND    : RECEIVED RESULTOF compute_cpi

FROM        : RULE_BOOK
TO          : DOAA ( BACK)
MSG_ID      : replyto_give_rule (status_eval_rule <RULE>)
MSG_TYPE    : reply

FROM        : DOAA
TO          : DOAA
MSG_ID      : compute_student_status ( stud_details <STRING>,
                                       status_eval_rule <RULE> )
PRE_COND    : RECEIVED replyto_give_rule
RESULT      : student_status <STRING>

FROM        : DOAA
TO          : DOAA
MSG_ID      : add_staus_to_grade_sheet (student_status <STRING>,
                                        stud_grade_sheet )
```

```
        PRE_COND   : RECEIVED RESULTOF compute_student_status

} ELSE { # all_couses_received == FALSE

        FROM       : DOAA
        TO         : DOAA_INTERFACE
        MSG_ID     : replyto_prepare_grade_sheets (remarks <STRING>,
                     course_grades_not_received <TABLE[c_no <STRING>]>)
                     # remarks = "Following course_coordinators have not
                     # send grade lists"
        MSG_TYPE   : reply

        FOREACH course_no (course_grades_not_received) {

            FROM       : DOAA
            TO         : COURSE (COURSE::c_no == $1~course_no)
            MSG_ID     : inform_instructors_to_send_grades
                         (msg_str = "Please send grades" <STRING>)
                         # "instructors" should be the name of the relation
                         # (declared in "RELATED WITH" field of COURSE class)
                         # with which the COURSE knows its co-ordinator


            FROM       : COURSE
            TO         : COURSE_COORDINATOR (ALL_RELATED)
            MSG_ID     : urgent_message (msg_str = "Please send grades"
                                         <STRING>)
            PRE_COND   : RECEIVED urgent_message

            FROM       : COURSE_COORDINATOR
            TO         : COURSE_COORD_INTERFACE
```

```
            MSG_ID     : urgent_message (msg_str <STRING>)
            PRE_COND   : RECEIVED inform_instructors_to_send_grades

        }

}

SUB_PROCESS :: get_student_details
#--------------------------------#

FROM       : COURSE
TO         : STUDENT (ALL_RELATED) # All STUDENT objects Related to this
                                   # COURSE object
MSG_ID     : give_details ($Evaluation_process!give_registered_students!query <ST
RESULT     : res {name <STRING>, roll_no <NUMBER>}

FROM       : STUDENT
TO         : COURSE ( BACK)
MSG_ID     : replyto_give_details ($name <STRING>, $roll_no <NUMBER>)
MSG_TYPE   : reply
ACTION     : SEND MESSAGE build_student_details_table

FROM       : COURSE
TO         : COURSE
MSG_ID     : build_stud_details_table
             ($give_details=name <STRING>, $give_details=roll_no <NUMBER>)
PRE_COND   : RECEIVED RESULTOF give_details
RESULT     : registered_students <TABLE [name <STRING>, roll_no <NUMBER>]>

FROM       : COURSE
TO         : COURSE
```

```
MSG_ID     : check_whether_all_details_received ()
RESULT     : all_details_received <BOOLEAN>

IF (all_details_reeived == TRUE) {

    FROM       : COURSE
    TO         : COURSE_COORDINATOR (BACK)
    MSG_ID     : replyto_send_registered_students
                 ( registered_students < TABLE [ student <STRING>,
                   roll_no<NUMBER>] >)
    MSG_TYPE   : reply

}

SUB_PROCESS :: get_details_from_student
#--------------------------------------#

FROM     : DOAA
TO       : STUDENT
MSG_ID   : give_cpi_spi()
ACTION   : SEND MESSAGE give_details
RESULT   : res{cpi<NUMBER>, spi<NUMBER>}


FROM     : STUDENT
TO       : PERFORMANCE_RECORD
MSG_ID   : give_details()
RESULT   : res{cpi<NUMBER>,spi<NUMBER>}

FROM     : STUDENT
```

```
TO       : DOAA
MSG_ID   : replyto_give_cpi_spi ( cpi<NUMBER>, spi<NUMBER> )
MSG_TYPE: reply


SUB_PROCESS :: check_student_for_warning
#---------------------------------------#

FROM     : DOAA
TO       : DOAA
MSG_ID   : check_for_warning ( cpi<NUMBER>, spi<NUMBER> )
PRE_COND: SPI < "2.0" && SPI <= "4.5" || CPI >= "5.0"
ACTION   : SEND MESSAGE give_warning_message OF check_student_for_warning


FROM : DOAA
TO     : STUDENT
MSG_ID : give_warning_message( remarks = " YOU ARE ON WARNING" < STRING> )

SUB_PROCESS :: check_student_on_acad_prob
#---------------------------------------#

FROM     : DOAA
TO       : DOAA
MSG_ID   : check_student_for_acad_prob ( cpi<NUMBER>, spi<NUMBER> )
PRE_COND: SPI < "4.5" && CPI < "5.0"
ACTION   : SEND MESSAGE give_acad_prob_msg OF check_student_on_acad_prob

FROM   : DOAA
TO     : STUDENT
MSG_ID : give_acad_prob_msg( remarks = "YOU ARE ON ACAD_PROB"<STRING>)
```

```
SUB_PROCESS :: check_for_dismissal
#---------------------------------#

FROM    : DOAA
TO      : DOAA
MSG_ID : check_student_for_dismissal()
PRE_COND: SPI <= "2.0" && CPI < "4.5"
ACTION : SEND MESSAGE give_dismissal_message OF check_for_dismissal

FROM : DOAA
TO    : STUDENT
MSG_ID : give_dismissal_message( remarks =" YOU ARE DISMISSED" <STRING>)

PROCESS :: Regestration_process
#-------------------------------#

FROM       : DOAA_INTERFACE
TO         : DOAA
MSG_ID     : prepare_reg_notice ()
ACTION     : DO GENERIC SUB_PROCESS make_notice #{ISSUEE : $TO,
                                    #res_message : reg_notice}
RESULT     : reg_notice <NOTICE>

FROM       : DOAA_INTERFACE
TO         : DOAA
MSG_ID     : send_reg_notice_to_depts (reg_notice <NOTICE>)
PRE_COND   : RECEIVED RESULTOF prepare_reg_notice
ACTION     : SEND MESSAGE notice_for_regestration

FROM       : DOAA
```

```
TO          : CONVENER (ALL)
MSG_ID      : notice_for_regestration (reg_notice <NOTICE>)
ACTION      : DO SUB_PROCESS store_notice

FROM        : STUDENT_INTERFACE
TO          : STUDENT
MSG_ID      : get_reg_form ()
ACTION      : SEND MESSAGE req_for_reg_form #{FROM:STUDENT, TO:CONVENER}

FROM        : STUDENT
TO          : CONVENER
MSG_ID      : req_for_reg_form ()
RESULT      : res_val {reg_form <REGESTRATION_FORM>, remarks <STRING>}
ACTION      : DO SUB_PROCESS check_eligibility_for_reg

# sending message to abstract class 'CONVENER' means sending to one of the
# derived classes
   :

FROM        : STUDENT
TO          : REGESTRATION_FORM
MSG_ID      : fill_form (values <STRING>)
MSG_TYPE    : exclusive # this type takes care of security

FROM        : STUDENT
TO          : CONVENER
MSG_ID      : sign_reg_form (reg_form <REGESTRATION_FORM>)
RESULT      : reg_form <REGESTRATION_FORM> # signed regestration form
ACTION      : DO SUB_PROCESS check_and_sign_reg_form
                             (reg_form <REGESTRATION_FORM>)

FROM        : STUDENT
```

```
TO          : DOAA
MSG_ID      : req_for_reg (reg_form <REGESTRATION_FORM>)
RESULT      : remarks <STRING>
ACTION      : DO SUB_PROCESS process_the_reg_form

#-------------------------------------------------------------------------#
PROCESS :: Evaluation_process
#---------------------------#

# course coordinator gets the students list for the course he teaches

FROM        : COURSE_COORD_INTERFACE
TO          : COURSE_COORDINATOR
MSG_ID      : get_registered_students ( course_no <STRING>)
RESULT      : registered_students <TABLE [student <STRING>, roll_no <NUMBER>]>
ACTION      : SEND MESSAGE give_registered_students

FROM        : COURSE_COORDINATOR
TO          : COURSE (c_no == course_no)
MSG_ID      : give_registered_students ( query = "name,roll_no" <STRING> )
RESULT      : registered_students < TABLE [ student <STRING>, roll_no <NUMBER>]>
PRE_COND    : RECEIVED get_registered_students
ACTION      : DO SUB_PROCESS get_student_details (query <STRING>)

# The semantics of these kind of TO constraints is  first check among
# the relatives whether a relation satisfying this constraint exists, if
# no, then send the message to class, which will resolve the constraint and
# send message to corresponding object(s)

FROM        : COURSE_COORDINATOR
TO          : COURSE_COOTD_INTERFACE ( BACK)
```

```
MSG_ID     : replyto_get_registered_students (registered_students < TABLE [
                                              student <STRING>, roll_no<NUMBER>]
MSG_TYPE   : reply

FROM       : COURSE_COORD_INTERFACE
TO         : COURSE_COORDINATOR
MSG_ID     : change_grade_list ( grade_list <TABLE
             [ student<STRING>, roll_no <NUMBER>, grade <CHARACTER>]>)
MSG_TYPE   : exclusive
ACTION     : DO SUB_PROCESS store_grade_list

FROM       : COURSE_COORD_INTERFACE
TO         : COURSE_COORDINATOR
MSG_ID     : send_grade_list_to_doaa ( grade_list
               < TABLE [student <STRING>, stud_rollno <NUMBER>,
                        grade <CHAR>]> )
POST_COND  : DONE SUB_PROCESS store_grade_list
ACTION     : SEND MESSAGE grade_list OF Evaluation_process

FROM       : COURSE_COORDINATOR
TO         : DOAA
MSG_ID     : grade_list ( grade_list < TABLE [ stud_rollno <NUMBER>,
                                    grade <CHAR>] >)
PRE_COND   : RECEIVED send_grade_list_to_doaa
ACTION     : DO SUB_PROCESS store_grade_list

FROM       : DOAA_INTERFACE
TO         : DOAA
MSG_ID     : prepare_grade_sheets ( )
ACTION     : DO SUB_PROCESS prepare_studs_grade_sheets
```

```
FROM       : DOAA_INTERFACE
TO         : DOAA
MSG_ID     : dispatch_grade_sheets ()
MSG_TYPE   : exclusive
ACTION     : SEND MESSAGE students_grade_sheets

FROM       : DOAA
TO         : DEPARTMENT
MSG_ID     : students_grade_sheets ( grade_sheets <GRADE_SHEET> )
PRE_COND   : DONE SUB_PROCESS prepare_studs_grade_sheets

FROM       : DEPARTMENT
TO         : STUDENT
MSG_ID     : grade_sheet ( grade_sheet <GRADE_SHEET> )
ACTION     : DO SUB_PROCESS store_grade_sheet
PRE_COND   : RECEIVED students_grade_sheets

PROCESS:: academic_probation
#----------------------------#

FROM  : DOAA
TO    : DOAA
MSG_ID: get_cpi_spi ( roll_no < NUMBER>)
ACTION: DO SUB_PROCESS get_details_from_student
RESULT: res{cpi<NUMBER>,spi<NUMBER>}
#-------------------------------------------------
GENERIC MESSAGE ::
#----------------------------#

# Calender queries
```

```
FROM                    : *
TO                      : CALENDER
MSG_ID                  : give_calender ()
RESULT                  : calender <CALENDER>

FROM                    : *
TO                      : CALENDER
MSG_ID                  : give_event (date <DATE>)
RESULT                  : event <STRING>

FROM                    : *
TO                      : CALENDER
MSG_ID                  : give_date (event <STRING>)
RESULT                  : date <DATE>

# Notice queries

FROM                    : *
TO                      : NOTICE_BOARD
MSG_ID                  : give_notice (constraints <STRING>)
# constraints can be 'date=xxx', issued_by=xxx, ALL  etc
RESULT                  : notice <NOTICE>

#-------------------------------------------------------------------------
```

Bibliography

[Boo94] Grady Booch. *Object Oriented Design and Applications.* Benjamin Cummins, 1994.

[R+91] James Rumbaugh et al. *Object Oriented Modelling and Design.* Prentice Hall, 1991.

[Raf96] D Rafee. Translating message centric oo specification to c++. Master's thesis, Dept. of CSE, Indian Institute of Technology, Kanpur, March 1996.

[Sar96] B Ramanjaneya Sarma. A message based specification system for object oriented software construction. Master's thesis, Dept. of CSE, Indian Institute of Technology, Kanpur, February 1996.

[Sri97] D S R Srinivasulu. An interpreter for a message centric specification system. Master's thesis, Dept. of CSE, Indian Institute of Technology, Kanpur, April 1997.

[Str91] Bjarne Stroustrup. *The C++ Programming Language.* Addison-Wesley, 2 edition, 1991.

[Vij96] M Vijayanand. Translating message centric oo specification to rdbms. Master's thesis, Dept. of CSE, Indian Institute of Technology, Kanpur, March 1996.

CSE-1997-M-SHR-APP